Abstract

Normalizing flows model complex probability distributions by combining a base distribution with a series of bijective neural networks. State-of-the-art architectures rely on coupling and autoregressive transformations to lift up invertible functions from scalars to vectors. In this work, we revisit these transformations as probabilistic graphical models, showing they reduce to Bayesian networks with a pre-defined topology and a learnable density at each node. From this new perspective, we propose the graphical normalizing flow, a new invertible transformation with either a prescribed or a learnable graphical structure. This model provides a promising way to inject domain knowledge into normalizing flows while preserving both the interpretability of Bayesian networks and the representation capacity of normalizing flows. We show that graphical conditioners discover relevant graph structure when we cannot hypothesize it. In addition, we analyze the effect of $\ell_1$-penalization on the recovered structure and on the quality of the resulting density estimation. Finally, we show that graphical conditioners lead to competitive white box density estimators. Our implementation is available at https://github.com/AWehenkel/DAG-NF.

1 Introduction

Normalizing flows [NFs, Rezende and Mohamed, 2015, Tabak et al., 2010, Tabak and Turner, 2013, Rippe and Adams, 2013] have proven to be an effective way to model complex data distributions with neural networks. These models map data points to latent variables through an invertible function while keeping track of the change of density caused by the transformation. In contrast to variational auto-encoders (VAEs) and generative adversarial networks (GANs), NFs provide access to the exact likelihood of the model’s parameters, hence offering a sound and direct way to optimize the network parameters. Normalizing flows have proven to be of practical interest as demonstrated by Van Den Oord et al. [2018], Kim et al. [2018] and Prenger et al. [2019] for speech synthesis, by Rezende and Mohamed [2015], Kingma et al. [2016] and Van Den Berg et al. [2018] for variational inference or by Papamakarios et al. [2019b] and Greenberg et al. [2019] for simulation-based inference. Yet, their usage as a base component of the machine learning toolbox is still limited in comparison to GANs or VAEs. Recent efforts have been made by Papamakarios et al. [2019a] and Kobyzev et al. [2020] to define the fundamental principles of flow design and by Durkan et al. [2019] to provide coding tools for modular implementations. We argue that normalizing flows would gain in popularity by offering stronger inductive bias as well as more interpretability.

Sometimes forgotten in favor of more data oriented methods, probabilistic graphical models (PGMs) have been popular for modeling complex data distributions while being relatively simple to build and read [Koller and Friedman, 2009, Johnson et al., 2016]. Among PGMs, Bayesian networks [BNs, Pearl and Russell, 2011] offer an appealing balance between modeling capacity and simplicity. Most notably, these models have been at the basis of expert systems before the big data era (e.g. [Díez et al., 1997, Kahn et al., 1997, Seixas et al., 2014]) and were commonly used to merge qualitative expert knowledge and quantitative information together. On the one hand, experts stated independence assumptions that should be encoded by the structure of the network. On the other hand, data were used to estimate the parameters of the conditional probabilities/densities encoding the quantitative aspects of the data distribution. These models have progressively received less attention from the machine learning community in favor of other methods.
that scale better with the dimensionality of the data.

Driven by the objective of integrating intuition into normalizing flows and the proven relevance of BNs for combining qualitative and quantitative reasoning, we summarize our contributions as follows: (i) From the insight that coupling and autoregressive transformations can be reduced to Bayesian networks with a fixed topology, we introduce the more general graphical conditioner for normalizing flows, featuring either a prescribed or a learnable BN topology; (ii) We show that using a correct prescribed topology leads to improvements in the modeled density compared to autoregressive methods. When the topology is not known we observe that, with the right amount of $\ell_1$-penalization, graphical conditioners discover relevant relationships; (iii) In addition, we show that graphical normalizing flows perform well in a large variety of density estimation tasks compared to classical black-box flow architectures.

## 2 Background

**Bayesian networks** A Bayesian network is a directed acyclic graph (DAG) that represents independence assumptions between the components of a random vector. Formally, let $\mathbf{x} = [x_1, \ldots, x_d]^T \in \mathbb{R}^d$ be a random vector distributed under $p_\mathbf{x}$. A BN associated to $\mathbf{x}$ is a directed acyclic graph made of $d$ vertices representing the components $x_i$ of $\mathbf{x}$. In this kind of network, the absence of edges models conditional independence between groups of components through the concept of d-separation [Geiger et al., 1990]. A BN is a valid representation of a random vector $\mathbf{x}$ iff its density can be factorized as

$$p_\mathbf{x}(\mathbf{x}) = \prod_{i=1}^d p(x_i | \mathcal{P}_i),$$

where $\mathcal{P}_i = \{j : A_{i,j} = 1\}$ denotes the set of parents of the vertex $i$ and $A \in \{0, 1\}^{d \times d}$ is the adjacency matrix of the BN. As an example, Fig. 1a is a valid BN for any distribution over $\mathbf{x}$ because it does not state any independence and leads to a factorization that corresponds to the chain rule. However, in practice we seek for a sparse and valid BN which models most of the independence between the components of $\mathbf{x}$, leading to an efficient factorization of the modeled probability distribution. It is worth noting that making hypotheses on the graph structure is equivalent to assuming certain conditional independence between some of the vector’s components.

**Normalizing flows** A normalizing flow is defined as a sequence of invertible transformation steps $g_k : \mathbb{R}^d \rightarrow \mathbb{R}^d$ ($k = 1, \ldots, K$) composed together to create an expressive invertible mapping $g := g_1 \circ \cdots \circ g_K : \mathbb{R}^d \rightarrow \mathbb{R}^d$. This mapping can be used to perform density estimation, using $g(\cdot; \theta) : \mathbb{R}^d \rightarrow \mathbb{R}^d$ to map a sample $\mathbf{x} \in \mathbb{R}^d$ onto a latent vector $\mathbf{z} \in \mathbb{R}^d$ equipped with a prescribed density $p_\mathbf{z}$ such as an isotropic Normal. The transformation $g$ implicitly defines a density $p(\mathbf{x}; \theta)$ as given by the change of variables formula,

$$p(\mathbf{x}; \theta) = p_\mathbf{z}(g(\mathbf{x}; \theta)) \left| \det J_{g(\mathbf{x}; \theta)} \right|,$$

where $J_{g(\mathbf{x}; \theta)}$ is the Jacobian of $g(\mathbf{x}; \theta)$ with respect to $\mathbf{x}$. The resulting model is trained by maximizing the likelihood of the model’s parameters $\theta$ given the training dataset $\mathbf{X} = \{x_1, \ldots, x_N\}$. Unless needed, we will not distinguish between $g$ and $g_k$ for the rest of our discussion.

In general the steps $g$ can take any form as long as they define a bijective map. Here, we focus on a subclass of normalizing flows for which the steps can be mathematically described as

$$g(\mathbf{x}) = \begin{bmatrix} g^1(x_1; c^1(\mathbf{x})) & \cdots & g^d(x_d; c^d(\mathbf{x})) \end{bmatrix}^T,$$

where the $c^i$ are the conditioners which role is to constrain the structure of the Jacobian of $g$. The functions $g^i$, partially parameterized by their conditioner, must be invertible with respect to their input variable $x_i$. They are often referred to as transformers, however in this work we will use the term normalizers to avoid any confusion with attention-based transformer architectures.

The conditioners examined in this work can be combined with any normalizer. In particular, we consider affine and monotonic normalizers. An affine normalizer $g : \mathbb{R} \times \mathbb{R}^2 \rightarrow \mathbb{R}$ can be expressed as $g(x; m, s) = x \exp(s) + m$, where $m \in \mathbb{R}$ and $s \in \mathbb{R}$.
are computed by the conditioner. There exist multiple methods to parameterize monotonic normalizers [Huang et al., 2018, De Cao et al., 2020, Durkan et al., 2019, Jaini et al., 2019], but in this work we rely on Unconstrained Monotonic Neural Networks [UMNNs, Wehenkel and Louppe, 2019] which can be expressed as

\[
g(x; c) = \int_0^x f(t, c) dt + \beta(c),
\]

where \( c \in \mathbb{R}^{|c|} \) is an embedding made by the conditioner and \( f : \mathbb{R}^{|c|+1} \rightarrow \mathbb{R}^+ \) and \( \beta : \mathbb{R}^d \rightarrow \mathbb{R} \) are two neural networks respectively with a strictly positive scalar output and a real scalar output. Huang et al. [2018] proved NFS built with autoregressive conditioners and monotonic normalizers are universal density approximators of continuous random variables.

### 3 Normalizing flows as Bayesian networks

**Autoregressive conditioners** Due to computing speed considerations, NFS are usually composed of transformations for which the determinant of the Jacobian can be computed efficiently, as otherwise its evaluation would scale cubically with the input dimension. A common solution is to use autoregressive conditioners, i.e., such that

\[
c^i(x) = h^i\left([x_1 \ldots x_{i-1}]^T\right)
\]

are functions \( h^i \) of the first \( i - 1 \) components of \( x \). This particular form constrains the Jacobian of \( g \) to be lower triangular, which makes the computation of its determinant \( O(d) \).

For the multivariate density \( p(x; \theta) \) induced by \( g(x; \theta) \) and \( p(z) \), we can use the chain rule to express the joint probability of \( x \) as a product of \( d \) univariate conditional densities,

\[
p(x; \theta) = p(x_1; \theta) \prod_{i=2}^d p(x_i | x_1; \theta).
\]

When \( p(z) \) is a factored distribution \( p(z) = \prod_{i=1}^d p(z_i) \), we identify that each component \( z_i \) coupled with the corresponding functions \( g^i \) and embedding vectors \( c^i \) encode for the conditional \( p(x_i | x_1; \theta) \). Therefore, and as illustrated in Fig. 1a, autoregressive transformations can be seen as a way to model the conditional factors of a BN that does not state any independence but relies on a predefined node ordering. This becomes clear if we define \( \mathcal{P}_i = \{x_1, \ldots, x_{i-1}\} \) and compare (4) with (1).

The complexity of the conditional factors strongly depends on the ordering of the vector components. While not hurting the universal representation capacity of normalizing flows, the arbitrary ordering used in autoregressive transformations leads to poor inductive bias and to factors that are most of the time difficult to learn. In practice, one often alleviates the arbitrariness of the ordering by stacking multiple autoregressive transformations combined with random permutations on top of each other.

**Coupling conditioners** Coupling layers [Dinh et al., 2015] are another popular type of conditioners that lead to a bipartite structure. The conditioners \( c^i \) made from coupling layers are defined as

\[
c^i(x) = \begin{cases} 
    h^i & \text{if } i \leq k \\
    h^i\left([x_1 \ldots x_k]^T\right) & \text{if } i > k
\end{cases}
\]

where the underlined \( h^i \in \mathbb{R}^{|c|} \) denote constant values and \( k \in \{1, \ldots, d\} \) is a hyper-parameter usually set to \( \left\lceil \frac{d}{2} \right\rceil \). As for autoregressive conditioners, the Jacobian of \( g \) made of coupling layers is lower triangular. Again, and as shown in Fig. 1b and 1c, these transformations can be seen as a specific class of BN where \( \mathcal{P}_i = \{\} \) for \( i \leq k \) and \( \mathcal{P}_i = \{1, \ldots, k\} \) for \( i > k \). D-separation can be used to read off the independencies stated by this class of BNs such as the conditional independence between each pair in \( x_{k+1:d} \) knowing \( x_1:k \). For this reason, and in contrast to autoregressive transformations, coupling layers are not by themselves universal density approximators even when associated with very expressive normalizers \( g^i \) [Wehenkel and Louppe, 2020]. In practice, these bipartite structural independencies can be relaxed by stacking multiple layers, and may even recover an autoregressive structure. They also lead to useful inductive bias, such as in the multi-scale architecture with checkerboard masking [Dinh et al., 2017, Kingma and Dhariwal, 2018].

#### Algorithm 1 Sampling

1. \( z \sim \mathcal{N}(0, I) \)
2. \( x \leftarrow z \)
3. repeat
   4. \( c^i \leftarrow h^i(x \circ A_i) \forall i \in \{1, \ldots, d\} \)
   5. \( x_i \leftarrow (g^i)^{-1}(z_i; c^i, \theta) \forall i \in \{1, \ldots, d\} \)
   6. until \( x \) converged

### 4 Graphical normalizing flow

#### 4.1 Graphical conditioners

Following up on the previous discussion, we introduce the graphical conditioner architecture. We motivate our approach by observing that the topological ordering (a.k.a. ancestral ordering) of any BN leads to a
lower triangular adjacency matrix whose determinant is equal to the product of its diagonal terms (proof in Appendix B). Therefore, conditioning factors \(c^i(x)\) selected by following a BN adjacency matrix necessarily lead to a transformation \(g\) whose Jacobian determinant remains efficient to compute.

Formally, given a BN with adjacency matrix \(A \in \{0,1\}^{d \times d}\), we define the **graphical conditioner** as being

\[
c^i(x) = h^i(x \odot A_{i,:}),
\]

where \(x \odot A_{i,:}\) is the element-wise product between the vector \(x\) and the \(i\)th row of \(A\) - i.e., the binary vector \(A_{i,:}\) is used to mask on \(x\). NFs built with this new conditioner architecture can be inverted by sequentially inverting each component in the topological ordering. In our implementation the neural networks modeling the \(h^i\) functions are shared to save memory and they take an additional input that one-hot encodes the value \(i\). An alternative approach would be to use a masking scheme similar to what is done by Germain et al. [2015] in MADE as suggested by Lachapelle et al. [2019].

The graphical conditioner architecture can be used to learn the conditional factors in a continuous BN while elegantly setting structural independencies prescribed from domain knowledge. In addition, the inverse of NFs built with graphical conditioners is a simple as it is for autoregressive and coupling conditioners, Algorithm 1 describes an inversion procedure. We also now note how these two conditioners are just special cases in which the adjacency matrix reflects the classes of BNs discussed in Section 3.

### 4.2 Learning the topology

In many cases, defining the whole structure of a BN is not possible due to a lack of knowledge about the problem at hand. Fortunately, not only is the density at each node learnable, but also the DAG structure itself: defining an arbitrary topology and ordering, as it is implicitly the case for autoregressive and coupling conditioners, is not necessary.

Building upon **Non-combinatorial Optimization via Trace Exponential and Augmented lagRangian for Structure learning** [NO TEARS, Zheng et al., 2018], we convert the combinatorial optimization of score-based learning of a DAG into a continuous optimization by relaxing the domain of \(A\) to real numbers instead of binary values. That is,

\[
\begin{align*}
\max_{A \in \mathbb{R}^{d \times d}} F(A) & \quad \Leftrightarrow \quad \max_{A \in \mathbb{R}^{d \times d}} F(A) \\
\text{s.t. } G(A) \in \text{DAGs} & \quad \text{s.t. } w(A) = 0,
\end{align*}
\]

where \(G(A)\) is the graph induced by the weighted adjacency matrix \(A\) and \(F : \mathbb{R}^{d \times d} \rightarrow \mathbb{R}\) is the log-likelihood of the graphical NF \(g\) plus a regularization term, i.e.,

\[
F(A) = \sum_{j=1}^{N} \log \left( p(x^j; \theta) \right) + \lambda_1 \|A\|_1,
\]

where \(\lambda_1\) is an \(\ell_1\)-regularization coefficient and \(N\) is the number of training samples \(x^i\). The likelihood is computed as

\[
p(x; \theta) = p_x(g(x; \theta)) \prod_{i=1}^{d} \frac{\partial g_i^k(x_i; h^i(x \odot A_{i,:}, \theta))}{\partial x_i}.
\]

The function \(w(A)\) that enforces the acyclicity is expressed as suggested by Yu et al. [2019] as

\[
w(A) := \text{tr} \left( (I + \alpha A)^d \right) - d \propto \text{tr} \left( \sum_{k=1}^{d} \alpha^k A^{k} \right),
\]

where \(\alpha \in \mathbb{R}_+\) is a hyper-parameter that avoids exploding values for \(w(A)\). In the case of positively valued \(A\), an element \((i,j)\) of \(A^k = \underbrace{A A \ldots A}_{k \text{ terms}}\) is non-null if and only if there exists a path going from node \(j\) to node \(i\) that is made of exactly \(k\) edges. Intuitively \(w(A)\) expresses to which extent the graph is cyclic. Indeed, the diagonal elements \((i,i)\) of \(A^k\) will be as large as there are many paths made of edges that correspond to large values in \(A\) from a node to itself in \(k\) steps.

In comparison to our work, Zheng et al. [2018] use a quadratic loss on the corresponding linear structural equation model (SEM) as the score function \(F(A)\). By attaching normalizing flows to topology learning, our method has a continuously adjustable level of complexity and does not make any strong assumptions on the form of the conditional factors.

### 4.3 Stochastic adjacency matrix

In order to learn the BN topology from the data, the adjacency matrix must be relaxed to contain reals instead of booleans. It also implies that the graph induced by \(A\) does not formally define a DAG during training. Work using NO TEARS to perform topology learning directly plug the real matrix \(A\) in (6) [Zheng et al., 2018; Yu et al., 2019, Lachapelle et al., 2019, Zheng et al., 2020] however this is inadequate because the quantity of information going from node \(j\) to node \(i\) does not continuously relate to the value of \(A_{i,j}\). Either the information is null if \(A_{i,j} = 0\) or it passes completely if not. Instead, we propose to build the stochastic pseudo binary valued matrix \(A'\) from \(A\),

\[
\text{max}_{A \in \mathbb{R}^{d \times d}} F(A) \quad \Leftrightarrow \quad \max_{A' \in \mathbb{R}^{d \times d}} F(A')
\]

\[
\text{s.t. } G(A) \in \text{DAGs} \quad \text{s.t. } w(A') = 0,
\]
defined as
\[
A_{i,j}' = \frac{e^{\log(\sigma(A_{i,j}^2)) + \gamma_1}}{e^{\log(\sigma(A_{i,j}^2)) + \gamma_1} + e^{\log(1 - \sigma(A_{i,j}^2)) + \gamma_2}},
\]
where \( \gamma_1, \gamma_2 \sim \text{Gumbel}(0,1) \) and \( \sigma(a) = 2(\text{sigmoid}(2a^2) - \frac{1}{2}) \) normalizes the values of \( A \) between 0 and 1, being close to 1 for large values and close to zero for values close to 0. The hyper-parameter \( T \) controls the sampling temperature and is fixed to 0.5 in all our experiments. In contrast to directly using the matrix \( A \), this stochastic transformation referred to as the Gumbel-Softmax trick in the literature [Maddison et al., 2016, Jang et al., 2016] allows to create a direct and continuously differentiable relationship between the weights of the edges and the quantity of information that can transit between two nodes. Indeed, the probability mass of the random variables \( A_{i,j}' \) is mainly located around 0 and 1, and its expected value converges to 1 when \( A_{i,j} \) increases.

4.4 Optimization

We rely on the augmented Lagrangian approach to solve the constrained optimization problem (6) as initially proposed by Zheng et al. [2018]. This optimization procedure requires solving iteratively the following sub-problems:

\[
\max_A \mathbb{E}_{\gamma_1, \gamma_2} [F(A)] - \lambda_t w(A) - \frac{\mu_t}{2} w(A)^2,
\]

where \( \lambda_t \) and \( \mu_t \) respectively denote the Lagrangian multiplier and penalty coefficients of the sub-problem \( t \).

We solve these optimization problems with mini-batch stochastic gradient ascent. We update the values of \( \gamma_t \) and \( \mu_t \) as suggested by Yu et al. [2019] when the validation loss does not improve for 10 consecutive epochs. Once \( w(A) \) equals 0, the adjacency matrix is acyclic up to numerical errors. We recover an exact DAG by thresholding the elements of \( A \) while checking for acyclicity with a path finding algorithm. We provide additional details about the optimization procedure used in our experiments in Appendix A.

5 Experiments

In this section, we demonstrate some applications of graphical NMs in addition to unifying NMs and BN under a common framework. We first demonstrate how pre-loading a known or hypothesized DAG structure can help finding an accurate distribution of the data. Then, we show that learning the graph topology leads to relevant BNs that support generalization.

Table 1: Datasets description. \( d \)=Dimension of the data. \( V \)=Number of edges in the ground truth Bayesian Network.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>( d )</th>
<th>( V )</th>
<th>Train</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic Circuit</td>
<td>8</td>
<td>10,000</td>
<td>5,000</td>
<td></td>
</tr>
<tr>
<td>8 Pairs</td>
<td>16</td>
<td>10,000</td>
<td>5,000</td>
<td></td>
</tr>
<tr>
<td>Tree</td>
<td>7</td>
<td>10,000</td>
<td>5,000</td>
<td></td>
</tr>
<tr>
<td>Protein</td>
<td>11</td>
<td>6,900</td>
<td>1,466</td>
<td></td>
</tr>
<tr>
<td>POWER</td>
<td>6</td>
<td>( \leq 15 )</td>
<td>1,659,917</td>
<td>204,928</td>
</tr>
<tr>
<td>GAS</td>
<td>8</td>
<td>( \leq 28 )</td>
<td>852,174</td>
<td>105,206</td>
</tr>
<tr>
<td>HEPMASS</td>
<td>21</td>
<td>( \leq 210 )</td>
<td>315,123</td>
<td>174,987</td>
</tr>
<tr>
<td>MINIBOONE</td>
<td>43</td>
<td>( \leq 903 )</td>
<td>29,556</td>
<td>3,648</td>
</tr>
<tr>
<td>BSDS300</td>
<td>63</td>
<td>( \leq 1,953 )</td>
<td>1,000,000</td>
<td>250,000</td>
</tr>
</tbody>
</table>

Table 2: Graphical vs autoregressive conditioners combined with monotonic normalizers. Average log-likelihood on test data over 5 runs, under-scripted error bars are equal to the standard deviation. Results are reported in nats; higher is better. The best performing architecture for each dataset is written in bold. Graphical conditioners clearly lead to improved density estimation when given a relevant prescribed topology in 3 out of the 4 datasets.

<table>
<thead>
<tr>
<th>Conditioner</th>
<th>Graphical</th>
<th>Autoreg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic Circuit</td>
<td>3.99 ( \pm ) 0.16</td>
<td>3.06 ( \pm ) 0.38</td>
</tr>
<tr>
<td>8 Pairs</td>
<td>(-0.44 \pm 0.06)</td>
<td>(-11.50 \pm 0.27)</td>
</tr>
<tr>
<td>Tree</td>
<td>(-6.86 \pm 0.02)</td>
<td>(-6.96 \pm 0.05)</td>
</tr>
<tr>
<td>Protein</td>
<td>6.46 ( \pm ) 0.08</td>
<td>7.52 ( \pm ) 0.10</td>
</tr>
</tbody>
</table>

well when combined with \( \ell_1 \)-penalization. Finally, we demonstrate that mono-step normalizing flows made of graphical conditioners are competitive density estimators.

5.1 On the importance of graph topology

The following experiments are performed on four distinct datasets, three of which are synthetic, such that we can define a ground-truth minimal Bayesian network, and the fourth is a causal protein-signaling network derived from single-cell data [Sachs et al., 2005]. Additional information about the datasets are provided in Table 1 and in Appendix C. For each experimental run we first randomly permute the features before training the model in order to compare autoregressive and graphical conditioners fairly.

Prescribed topology Rarely do real data come with their associated Bayesian network however oftentimes experts want to hypothesize a network topology and to rely on it for the downstream tasks. Sometimes the topology is known a priori, as an example the sequence of instructions in stochastic simulators can usually be translated into a graph topology (e.g. in probabilistic programming [van de Meent et al., 2018, Weilbach et al., 2020]). In both cases, graphical conditioners allow to explicitly take advantage of this to
build density estimators while keeping the assumptions made about the network topology valid.

Table 2 presents the test likelihood of autoregressive and graphical normalizing flows on the four datasets. The flows are made of a single step and use monotonic normalizers. The neural network architectures are all identical. Further details on the experimental settings as well as additional results for affine normalizers are respectively provided in Appendix C.2 and C.3. We observe how using correct BN structures lead to good test performance in Table 2. Surprisingly, the performance on the protein dataset are not improved when using the ground truth graph. We observed during our experiments (see Appendix C.3) that learning the topology from this dataset sometimes led to improved density estimation performance with respect to the ground truth graph. The limited dataset size does not allow us to answer if this comes from the limited capacity of the flow and/or from the erroneous assumptions in the ground truth graph. However, we stress out that the graphical flow respects the assumed causal structure in opposition to the autoregressive flow.

Learning the topology The $\lambda_1$ coefficient introduced in (7) controls the sparsity of the optimized BN, allowing to avoid some spurious connections. We now analyze the effect of sparsity on the density estimation performance. Fig. 2a shows the test log-likelihood as a function of the $\ell_1$-penalization on the 8 pairs dataset. We observe that the worst results are obtained when there is no penalization. Indeed, in this case the algorithm finds multiple spurious connections between independent vector’s components and then overfits on these incorrect relationships. Another extreme case shows up when the effect of penalization is too strong, in this case, the learning procedure underfits the data because it ignores too many relevant connections. It can also be concluded from the plot that the optimal $\ell_1$-penalization performs on par with the ground truth topology, and certainly better than the autoregressive conditioner. Additional results on the other datasets provided in Appendix C lead to similar conclusions.

**Protein network** The adjacency matrix discovered by optimizing (8) (with $\lambda_1 = 12$) on the protein network dataset is shown in Fig. 3. All the connections discovered by the method correspond to ground truth connections. Unsurprisingly, their orientation do not always match with what is expected by the experts. Overall we can see that the optimization procedure is able to find relevant connections between variables and avoids spurious ones when the $\ell_1$-penalization is optimized. Previous work on topology learning such as Zheng et al. [2018], Yu et al. [2019], Lachapelle et al. [2019] compare their method to others by looking at the topology discovered on the same dataset. Here we do not claim that learning the topology with a graphical conditioner improves over previous methods. Indeed, we believe that the difference between the methods mainly relies on the hypothesis and/or inductive biased made on the conditional densities. Which method would perform better is dependent on the application. Moreover, the Structural Hamiltonian Distance (SHD) and the Structural Inference Distance (SID), often used to compare BNs, are not well motivated in general. On the one hand the SHD does not take into account the independence relationships modeled with Bayesian networks, as an example the SHD between the graph found on 8 pairs dataset of Fig. 2b and one possible ground truth is non zero whereas the two BNs are equivalent. On the other hand the SID aims to compare causal graphs whereas the methods...
discussed here do not learn causal relationships. In general BN topology identification is an ill posed problem and thus we believe using these metrics to compare different methods without additional downstream context is irrelevant. However, we conclude from Fig. 2b and Fig. 3 that the scores computed with graphical NFs can be used to learn relevant BN structures.

### 5.2 Density estimation benchmark

In these experiments, we compare autoregressive, coupling and graphical conditioners with no $\ell_1$-penalization on benchmark tabular datasets as introduced by Papamakarios et al. [2017] for density estimation. See Table 1 for a description. We evaluate each conditioner in combination with monotonic and affine normalizers. We only compare NFs with a single transformation step because our focus is on the conditioner capacity. We observed during preliminary experiments that stacking multiple conditioners improves the performance slightly, however the gain is marginal compared to the loss of interpretability. To provide a fair comparison we have fixed in advance the neural architectures used to parameterize the normalizers and conditioners as well as the training parameters by taking inspiration from those used by Wehenkel and Louppe [2019] and Papamakarios et al. [2017]. The variable ordering of each dataset is randomly permuted at each run. All hyper-parameters are provided in Appendix D and a public implementation will be released on Github.

First, Table 3 presents the test log-likelihood obtained by each architecture. These results indicate that graphical conditioners offer the best performance in general. Unsurprisingly, coupling layers show the worst performance, due to the arbitrarily assumed independencies. Autoregressive and graphical conditioners show very similar performance for monotonic normalizers, the latter being slightly better on 4 out of the 5 datasets. Table 4 contextualizes the performance of graphical normalizing flows with respect to the most popular normalizing flow architectures. Comparing the results together, we see that while additional steps lead to noticeable improvements for affine normalizers (MAF), benefits are questionable for monotonic transformations. Overall, graphical normalizing flows made of a single transformation step are competitive with the best flow architectures with the added value that they can directly be translated into their equivalent BNs. From these results we stress out that single step graphical NFs are able to model complex densities on par with the most popular flow architectures whereas it is only made of 1 transformation step.

### 6 Discussion

**Cost of learning the graph structure** The attentive reader will notice that learning the topology does not come for free. Indeed, the Lagrangian for-
Table 5: Rounded average number of edges (over 3 runs) in the equivalent Bayesian network. The graphical conditioners lead to sparser BNs compared to autoregressive conditioners.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>P</th>
<th>G</th>
<th>H</th>
<th>M</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph.-Aff.</td>
<td>15</td>
<td>26</td>
<td>152</td>
<td>277</td>
<td>471</td>
</tr>
<tr>
<td>Graph.-Mon.</td>
<td>15</td>
<td>27</td>
<td>159</td>
<td>265</td>
<td>1594</td>
</tr>
<tr>
<td>Coupling</td>
<td>9</td>
<td>16</td>
<td>110</td>
<td>462</td>
<td>992</td>
</tr>
<tr>
<td>Autoreg.</td>
<td>15</td>
<td>28</td>
<td>210</td>
<td>903</td>
<td>1953</td>
</tr>
</tbody>
</table>

mulation requires solving a sequence of optimization problems which increases the number of epochs before convergence. In our experiments we observed different overheads depending on the problems, however in general the training time is at least doubled. This does not impede the practical interest of using graphical normalizing flows. The computation overhead is more striking for non-affine normalizers (e.g. UMNMs) that are computationally heavy. However, we observed that most of the time the topology recovered by affine graphical NFs is relevant. It can thus be used as a prescribed topology for normalizers that are heavier to run, hence alleviating the computation overhead. Moreover, one can always hypothesize on the graph topology but more importantly the graph learned is usually sparser than an autoregressive one while achieving similar if not better results. The sparsity is interesting for two reasons: it can be exploited for speeding up the forward density evaluation; but more importantly it usually corresponds to shallower BNs that can be inverted faster than autoregressive structures.

Bayesian network topology learning Formal BN topology learning has extensively been studied for more than 30 years now and many strong theoretical results on the computational complexity have been obtained. Most of these results however focus on discrete random variables, and how they generalize in the continuous case is yet to be explained. The topic of BN topology learning for discrete variables has been proven to be NP-hard by Chickering et al. [2004]. However, while some greedy algorithms exist, they do not lead in general to a minimal I-map although allowing for an efficient factorization of random discrete vectors distributions in most of the cases. These algorithms are usually separated between the constrained-based family such as the PC algorithm [Spirtes et al., 2001] or the incremental association Markov blanket [Koller and Friedman, 2009] and the score-based family as used in the present work. Finding the best BN topology for continuous variables has not been proven to be NP-hard however the results for discrete variables suggest that without strong assumptions on the function class the problem is hard.

The recent progress made in the continuous setting relies on the heuristic used in score-based methods. In particular, Zheng et al. [2018] showed that the acyclicity constraint required in BNs can be expressed with NO TEARS, as a continuous function of the adjacency matrix, allowing the Lagrangian formulation to be used. Yu et al. [2019] proposed DAG-GNN, a follow up work of Zheng et al. [2018] which relies on variational inference and auto-encoders to generalize the method to non-linear structural equation models. Further investigation of continuous DAG learning in the context of causal models was carried out by Lachapelle et al. [2019]. They use the adjacency matrix of the causal network as a mask over neural networks to design a score which is the log-likelihood of a parameterized normal distribution. The requirement to pre-define a parametric distribution before learning restricts the factors to simple conditional distributions. In contrast, our method combines the constraints given by the BN topology with NFs which are free-form universal density estimators. Remarkably, their method leads to an efficient one-pass computation of the joint density. This neural masking scheme can also be implemented for NF architectures such as already demonstrated by Papamakarios et al. [2017] and De Cao et al. [2019] for autoregressive conditioners.

Shuffling between transformation steps As already mentioned, consecutive transformation steps are often combined with randomly fixed permutations in order to mitigate the ordering problem. Linear flow steps [Oliva et al., 2018] and 1x1 invertible convolutions [Kingma and Dhariwal, 2018] generalize these fixed permutations. They are parameterized by a matrix \( W = PLU \) where \( P \) is the fixed permutation matrix, and \( L \) and \( U \) are respectively a lower and an upper triangular matrix. Although linear flow improves the simple permutation scheme, they do still rely on an arbitrary permutation. To the best of our knowledge, graphical conditioners are the first attempt to get completely rid of any fixed permutation in NFs.

Inductive bias Graphical conditioners eventually lead to binary masks that model the conditioning components of a factored joint distribution. In this way, the conditioners process their input as they would process the full vector. We show experimentally in Appendix E that this effectively leads to good inductive bias for processing images with NFs. In addition, we have shown that normalizing flows built from graphical conditioners combined with monotonic transformations are expressive density estimators. In effect, this means that enforcing some a priori known independencies can be performed thanks to graphical normal-
izing flows without hurting their modeling capacity. We believe such models could be of high practical interest because they cope well with large datasets and complex distributions while preserving some readability through their equivalent BN.

Close to our work, Weilbach et al. [2020] improve amortized inference by prescribing a BN structure between the latent and observed variables into a FFJORD NF, once again showing the interest of using the potential BN knowledge. Similar to our work, Khemakhem et al. [2020] see causal autoregressive flows as structural equation modelling. They show bivariate autoregressive affine flows can be used to identify the causal direction under mild conditions. Under similar mild conditions, discovering causal relationships with graphical normalizing flows could well be an exciting research direction.

Conclusion We have revisited coupling and autoregressive conditioners for normalizing flows as Bayesian networks. From this new perspective, we proposed the more general graphical conditioner architecture for normalizing flows. We have illustrated the importance of assuming or learning a relevant Bayesian network topology for density estimation. In addition, we have shown that this new architecture compares favorably with autoregressive and coupling conditioners and on par to the most common flow architectures on standard density estimation tasks even without any hypothesized topology. One interesting and straightforward extension of our work would be to combine it with normalizing flows for discrete variables. We also believe that graphical conditioners could be used when the equivalent Bayesian network is required for downstream tasks such as in causal reasoning.

Acknowledgments

We thank Vân Anh Huynh-Thu, Johann Brehmer, and Louis Wehenkel for proofreading this manuscript or an earlier version of it. We are also thankful to the reviewers for helpful comments. Antoine Wehenkel is a research fellow of the F.R.S.-FNRS (Belgium) and acknowledges its financial support. Gilles Louppe is recipient of the ULiège - NRB Chair on Big data and is thankful for the support of NRB.

References


Matthew J Johnson, David K Duvenaud, Alex Wiltschko, Ryan P Adams, and Sandeep R Datta.


Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. An introduction


